Chapter 3: A reactive scheme for fault recovery

3.1. Introduction

A realtime scheme as described in the previous chapter gives the application guarantees about the network performance that are only valid in the absence of network faults. Such a scheme is susceptible to link faults because of the connection-oriented nature of the network protocols, which dictates that the route followed by the packets during data transfer be pre-determined, so that resources on the links and nodes along the path can be reserved. This implies that, if any link on the route is affected by a fault, the flow of data is interrupted until and unless an alternate route can be found and the reservations transfered to it. Thus, we need to provide fault handling mechanisms to the realtime network. This chapter describes fault recovery mechanisms, which use reactive techniques to restore traffic in the event of a fault. Chapter 6 describes pro-active techniques that provide transparent tolerance to faults.

In the next section, we describe the goals of the fault recovery mechanisms, which drive the design and evaluation process. In Section 3.3, we discuss the assumptions about the network, derived from the basic underlying principles of realtime networks, on which we base the design of our fault recovery mechanisms.

We start the design process in Section 3.4 by providing a framework that consists of the basic tasks which any fault recovery scheme would have to perform. In Section 3.5, we consider some of the fundamental design issues, such as whether the scheme should be centralized or distributed, and whether we should attempt to find an optimal solution or use fast greedy algorithms. We make some fundamental design choices at this stage, and justify them on the basis of the principles of realtime networks. We also leave some of the fundamental issues open for investigation through simulation. In Section 3.6, we further narrow the solution space to be explored by choosing not to explore a number of issues. Finally, Section 3.7 identifies the design choices that remain to be explored. This leaves us with a well-structured problem domain, which we can systematically explore using simulations tools to be described in the next chapter.

3.2. Objectives of the fault recovery mechanisms

The fault recovery mechanisms are intended to provide fault handling that is efficient in the common fault-free situation. Accordingly, we will attempt to devise techniques that use no extra resources and cause no additional channel management activity in the absence of faults. When a fault does occur, some latency before service is restored to the affected realtime channels is acceptable. Of course, low latency is one of the objectives, and thus the basis of a metric, by which different schemes may be compared.

The recovery mechanisms are intended to restore realtime connectivity with the original performance guarantees to the affected channels. However, when a fault occurs, depending on the load in the network, all the affected channels may not be successfully rerouted. Rerouting as much traffic as possible is an important objective, and this defines another metric for comparison of recovery schemes. The effect of the rerouting process on the ability of the network to accept more channels immediately after the recovery process terminates is also a consideration. We would like the recovery process to use resources as efficiently as possible. Finally, we would like the recovery schemes to be as flexible as possible, to deal with different network topologies, load conditions, and traffic mixes. To summarize, the objectives of fault recovery are to:

- Use no extra resources during normal operation
- Restore original performance guarantees to channels
- Recover as much traffic as possible
- Minimize reaction time to the extent possible
- Use resources efficiently during rerouting
- Adapt flexibly to different topologies, load conditions, and traffic mixes

3.3. Network model and assumptions

The Tenet approach is one of the many approaches that can guarantee performance in a packet-switching network. Other solutions have been proposed by other investigators, some of which are mentioned in Chapter 2 of this thesis. All these schemes share some common features, which appear to be necessary to the provision of realtime guarantees on packet/cell switched networks. It has been argued (30, 32) that some fundamental assumptions must be satisfied in order to provide realtime services as defined here on packet/cell switched networks. If we can design our fault recovery schemes based on these fundamental assumptions about the network and the realtime scheme, then there is reason to believe that the fault recovery schemes will apply to any of the realtime schemes. These fundamental assumptions are listed below.

Resource reservation and admission control: Every network has some limit on such resources as bandwidth and buffers, and, as the offered load to the network increases, a point of congestion is eventually reached when the performance of the network deteriorates. In order to provide good performance to realtime applications even during periods of network congestion, it is necessary to set aside some fraction of the resources so that these resources are available to the realtime packets in preference to others. All the guaranteed performance schemes mentioned in Chapter 1 implement some form of resource reservation. If realtime traffic is added to the network, a point must eventually be reached when the addition of another stream could possibly lead to overload on the network, leading to violation of guarantees. While the algorithms or heuristics to detect this condition differ from scheme to scheme, all realtime schemes must employ some form of admission control.

Client traffic specification: Since the network cannot support an arbitrary amount of traffic, it is necessary for each realtime client to give the network a description of the traffic it will inject into the network. This traffic description is used to perform admission control, to make sure that the addition of a new client's traffic will not violate the network's ability to provide guaranteed service to all other clients, as well as to satisfy the new client's requirements. The client must adhere to these traffic descriptions, otherwise the traffic on the network will be higher than calculated, and realtime performance will suffer.

Connection-orientedness: The resources reserved for a realtime client, such as transmission bandwidth, buffers, and so on, are local in nature, that is, they exist in a given switch or router. Thus, the reservations must be made over a specific set of switches or routers on the network. This set of switches or routers would define one or more paths in the network, over which resources are reserved. During transmission, the packets of realtime data would follow these paths. These route(s) have to be computed before the transmission of data, in order that the resources be available when they are needed. Note that the route computation need not occur strictly before the establishment, since it may be done in a distributed manner as the establishment request makes its way through the network. However, before the data transfer starts, both processes have to be complete. Thus, the nature of the communication is connection-oriented.

Protective service discipline: The guarantees offered to a realtime client should be protected from disruption by misbehavior of other realtime or non-realtime traffic sources. This could be achieved by a variety of techniques, but a minimum seems to be providing priority of realtime traffic over non-realtime traffic, and some form of rate control or policing on each realtime source. The rate control ensures that the traffic introduced by a client into the network does not exceed its traffic specifications, thereby protecting other realtime clients. Various combinations of policing mechanisms and priority schemes have been proposed (26, 50, 71).

3.3.1. A general realtime networking model

For the purposes of designing our fault recovery schemes, we assume that the network meets the above set of requirements. We proceed to make this network model more concrete by describing the outlines of a scheme, which fills in some more details about the realtime network, but is still general enough to fit a number of the schemes described in Chapter 2 (3, 19, 20, 35, 38, 63, 68). This leads us to believe that this model will fit the schemes used to provide realtime switched virtual circuits (SVCs with QoS guarantees) on future ATM networks.

- The network handles a request for a realtime channel in a distributed fashion. We do not consider centralized approaches for reasons of scalability of the scheme.
- Channel establishment is based on a round-trip message exchange, during which admission control tests are performed at all the nodes on the route of the channel, based on local resource state information available at each node. The single round-trip paradigm is assumed because of its efficiency and compatibility with the virtual circuit model of ATM networks.
- The best performance possible is reserved on the forward pass. The resources are relaxed to the appropriate performance level on the reverse pass of the round trip; the necessary computations may be performed at the destination, or in a distributed fashion during the reverse pass. The two-pass scheme allows us to maximize the chances of acceptance within one round-trip time, yet does not waste resources within the network because of over-reservation.

Note that the above outline does not place any restrictions on the route selection process. It may be performed on the originator of the establishment request, at a route server, or in a distributed fashion as the establishment request moves through the network on its forward pass. The specifics of the admission control tests performed and the resources reserved are also unspecified, and depend on the particular scheme being used.

To allow route modification of realtime channels, we may either assume support from the realtime scheme, or incorporate the appropriate channel management functionality in the fault recovery protocol. We chose the former approach, since route modification is already supported in a version of the Tenet Suite 1 by the DCM protocol. Thus, we need to make another assumption about the realtime scheme.

- The realtime scheme supports modification of the route of the channel during its lifetime. Using a three-pass mechanism, the resources of the channel can be modified without violating any guarantees during the transition process. The first pass makes tentative reservations for the new route or performance parameters, the reverse pass relaxes the reservations and also makes them permanent. At this point data starts to follow the new path. The third pass removes the resources along the old route.

The above model of the realtime scheme leads to the following observation, which must also be kept in mind while designing routing and recovery algorithms.

- The realtime network has a fairly complex resource state, distributed across the nodes of the network, which must be considered while seeking a new route for a channel. The exact parameters of the resource state depend on the particular realtime scheme under consideration. But the information required to perform the admission control tests is also required by the routing algorithm, in order for it to find routes on which the channel establishment request will be likely not to fail.

In addition to the above assumptions, we also need to keep in mind the assumptions about the underlying network and the service model described in the previous chapter. Two of these are repeated below, because they are significant to the choices made in this chapter.

- The nodes of the network are programmable computers with sufficient computation power to run the channel management protocols with admission control tests. These same computers can also be used to run the routing algorithm, as long as the algorithm is of reasonable time complexity.
- The channel establishment and teardown requests are frequent. Thus, the set of channels, and hence the set of routes, existing in the network at any given time is very dynamic.

3.4. A fault recovery framework

In this section, we impose a framework on the fault recovery scheme by enumerating five tasks which must be performed in order to restore traffic in the event of a fault. These tasks are *detection*, *instigation* of recovery, *route selection*, *connection management*, and *return to normal*.

Detection is performed by the nodes adjacent to the failed component. The latency requirement dictates that the time from the occurrence of the fault to the detection must be small. On the other hand, the requirement that channel management activity during normal fault-free operation be kept to a minimum implies that faults should not be declared without some level of validation. We will not explore this task in any further detail in this chapter. Chapter 5 contains some considerations on the details of the detection mechanisms.

Instigation of the restoral process is performed by the node that detects the fault. This consists of informing the rest of the network of the fault so that recovery can be performed. The information that needs to be communicated includes the identity of the failed component(s), the affected channels, and, depending on the recovery scheme, possibly the new routes. We will look at this process in more detail in this chapter and in Chapter 5.

Route selection is one of the most important stages of recovery. In this stage, the network chooses the new set of routes, to which the channel management process will transfer the channels affected by the fault. This process may be completely centralized, so that the entire set of routes are computed on one computer. It may be partially distributed, so that each route is computed on a single computer, but different routes are computed by different computers. Each single route may also be computed in a distributed fashion, on a hop-by-hop basis as the establishment message works its way through the network. Finally, the order and timing of the route computations can also be varied.

Channel management is the process of actually moving the network resources from the old path to the new. Channel management attempts to reserve sufficient resources on the new path to meet the end-to-end performance requirements. If the attempt fails, we may return to the route selection process again, perhaps with additional constraints defined by the specific retry policy. If the attempt succeeds, we remove the remaining resources from the old path.

Return to normal is the action taken to balance the network load more evenly when the link is declared good by the detection and validation mechanisms. Since the longer routes used after rerouting consume more resources, the capacity of the network can be increased by returning the channels to shorter paths using the newly restored links. The important consideration here is not to affect the traffic during this process. The scheme should also work well with the admission control protocol. One possible return to normal scheme is described in Chapter 5.

3.5. Design issues for a reactive scheme for fault recovery

In the framework described above, the tasks that are of particular interest to us in this chapter are the processes of instigation, route selection, and channel management. We assume, for the moment, that detection and return to normal can be performed. These are discussed in more detail in Chapter 5.

In this section, we look at some of the general design choices that exist in designing a reactive scheme for fault recovery, in the context of the assumptions that we have made about the realtime network. We looked at some of these issues in Section 1.4.2, in the context of survivability for the cross-connect layer of telecommunication networks. However, the assumptions regarding realtime channels, which were described in Section 3.3, are different from the ones on the basis of telecommunication networks are designed.

For example, in the cross-connect layer of the telecommunication network, the possible lack of intelligence or computing power of the DCS was an important issue in favor of centralized schemes. Here, we can assume that the switches (or the controllers attached to the switches) will have sufficient computing power to run distributed algorithms of reasonable complexity, since we already need such processing ability to run the admission control protocols. The state information associated with a realtime channel is also larger than for a cross-connect trunk, since, instead of just bandwidth, each channel has several traffic and performance parameters associated with it. The network state at intermediate nodes, which summarizes the local performance bounds allocated to existing channels, also needs to be considered during rerouting, in order that the algorithm be able to compute routes on which sufficient resources exist to support the new channel. This network state is also much more dynamic than that of the cross-connect network, since realtime channels have shorter life-times than cross-connect trunks. We need to look at these fundamental design issues again, with these different assumptions in mind.

The questions that we will consider in this section are the following. Should the algorithms attempt to find optimal solutions or should they aim at fast approximate solutions using a greedy strategy? Should they be distributed or centralized? Should we use precomputation of routes to speed up the recovery or should the reroutes be computed dynamically when the fault occurs? How much time should we spend on information exchange during the recovery process? Some of these questions will be answered in this section, while others will be kept open for investigation through simulation.

3.5.1. Optimal vs. greedy

The problem of finding the optimal flow in a network given bandwidth constraints on the links can be mapped onto a linear programming problem, for which reasonably efficient algorithms (23,61) exist. The routing problem at the cross-connect layer of telecommunication networks can be approximated by the above model, and quasi-optimal solutions can be found efficiently. However, when the admission control test becomes more complex than checking a simple bandwidth constraint, especially with non-linear constraints used in the traffic specifications, as is the case for the X_{min}, X_{ave}, I model, the problem can no longer be simplified to a linear program. and no known efficient algorithm for computing an optimal solution exists. The brute force algorithm, which enumerates all possible solutions, evaluates them in terms of some criteria of success, and picks the best, is exponential in time complexity. This makes such schemes infeasible for large graphs or if the state information per node, on the basis of which the optimal solution is chosen, is complex, as is the case for realtime networks.

Greedy algorithms are a class of algorithms that deal with problems involving many similar steps one at a time, finding the best incremental solution for each. In the context of establishing realtime channels, this implies dealing with channels one at a time and picking a route placement and resource allocation that minimizes the increase in resource usage. Greedy algorithms may produce non-optimal solutions. However, since each application of the algorithm attempts to minimize the increase in resource consumption caused by the single step, the overall solution tends to stay close to the optimal. Greedy algorithms provide reasonably good, fast solutions to otherwise computationally intractable problems. The greedy approach is also used in our network model for channel establishment (see Ch. 2), i.e., the routing and resource allocation of one channel does not involve making adjustments to the route or resource allocation of any other channel. This approach is much faster, since the time complexity of the resource allocation is now linear in the number of channels and nodes, and no channel management activity to move existing channels is required. This is especially important considering the dynamicity of the realtime load on the network, since we either have to optimize every time a new realtime channel is added or removed, or settle for a non-optimal solution. We will explore a greedy approach to fault recovery, because it fits well with our realtime network model, operates in linear time, and produces reasonably good solutions.

3.5.2. Centralized vs. distributed

A centralized scheme gathers all the important network state at one computer, where a new configuration for the network is computed. Since all the information is available, it is possible to compute a configuration that is optimal according to some target function. This optimal scheme might be able to reroute more channels successfully under heavy-load conditions. However, this gain comes at the expense of a possibly higher latency, since the time to gather the network state to one location and then distribute the network configuration to the switches must be added to the time to run the algorithm. The time to gather the network state has to be considered here, since, unlike the cross-connect networks, realtime networks have a very dynamic load, with connections coming and going as applications are started and ended. Since the establishment protocols are distributed, no single computer has the complete network configuration at the time of the fault. All the network state must be gathered to the central computer before the centralized algorithm can start operating. This latency would be negligible for a LAN, but might be significant for a WAN. Furthermore, this higher latency would translate to a much higher number of bits lost in a network with a high bandwidth-delay product.

Centralized algorithms are susceptible to single-point failures, since all recoveries would be stalled by a failure of the central computer, or of the control network which connects the central computer to the switches. In the telecommunication networks, this problem is mitigated by having more than one fault recovery computer and having redundancy in the control network, so that the control network itself is fault-tolerant. However, this increases the cost of the solution. We do not assume a separate control network; rather, the recovery action is carried out over the data network itself. This fits in better with our model of the realtime establishment protocols, which exchange control messages over the data network. However, this makes the centralized approach much less desirable, since the failures in the network that we need to recover from make the collection and distribution process itself unreliable. The distributed algorithms deal explicitly with the problem of failed links. For example, the exchange of control messages with neighbors may be used as part of the process to determine the working topology of the network, since the inability to exchange messages indicates a failure of the corresponding link.

A distributed algorithm also scales better to larger networks, since the processing power available increases with the network size. The centralized scheme, on the other hand, must run an optimization algorithm of exponential complexity on a single computer. Thus, the major advantage of using a centralized solution (i.e., finding an optimal solution) is infeasible for all but the smallest realtime networks. The establishment protocol assumed in the network model is distributed for much the same reasons. A distributed fault recovery scheme has the advantage of fitting better with our distributed network model.

A distributed algorithm would find a non-optimal solution based on a limited exchange of information faster than a centralized scheme and without relying on a single central computer. This comes at the expense of possibly finding a solution considerably worse than the optimal. Since the distributed elements may have inconsistent views of the network, the overall behavior of the system may be hard to understand for an implementor, and may be far from the expected and desirable behavior, even though each individual element has a simple and understandable behavior. In spite of these disadvantages, this approach appears more feasible for the case of realtime networks, and will be explored here.

The channel management activity described in Chapter 2 is completely distributed, since some admission control tests are performed on each node in the path of a candidate route. Selection of a candidate route may be either partially or completely distributed. For a partially distributed route selection, each channel is routed separately on a separate computer, but each individual route computation takes place on a single computer. For a completely distributed scheme, each route is computed hop-by-hop in a distributed manner as the establishment request progresses through its forward pass. While a fully distributed scheme is worth investigating, the partially distributed route selection strategy is known to work well (it is used in the basic channel management protocol), and will be used here.

3.5.3. Pre-computed vs. dynamic

The task of rerouting may be partially pre-computed in order to decrease the response time of the recovery after a failure. This idea has been explored in the context of the cross-connect layer of telecommunication networks, as mentioned in Section 1.4.2. In the context of realtime networks, multimedia traffic is expected to be much more dynamic than the configuration presented by the cross-connect connections. A trunk is a relatively long-lived connection. As such, the requirements of a set of trunks, characterized by source, destination, and bandwidth, are fairly static. After the set of required trunks has been decided, the current configuration, as well as the recovery configurations for all likely fault combinations, may be computed and stored. On the other hand, for a realtime network, the set of active connections changes much more dynamically. In addition, the time and space complexity of computing and storing the recovery configurations for any such set is also larger, since the channel has a more complex state than a cross-connect trunk. This makes it infeasible to pre-compute the set of recovery configurations for all possible fault conditions each time the network state changes. A dynamic approach, which computes the reconfiguration when the fault occurs, will be developed and explored here.

3.5.4. Global vs. local knowledge

The extent to which knowledge is universally shared among the elements of a distributed algorithm can have a tremendous impact on the performance and correctness of the algorithm. At one extreme, we might allow all nodes to gain complete information about any change in the network state before computing the next step in the algorithm. Since this would imply that all nodes have completely up-to-date knowledge of the network state throughout the reconfiguration process, this is equivalent to a centralized algorithm in terms of goodness or optimality of the solution. At the other extreme, we may choose to allow all nodes to run their reconfiguration algorithms with the view of the network state that they had at the moment of the fault, without allowing any additional communication. This would have the fastest response time of any algorithm, but would reroute fewer connections successfully, since the inconsistencies among the network views of the separate network elements would lead to poor cooperation between the nodes. Thus, by allowing more time for communication, we can improve the success of the rerouting algorithm.[1]

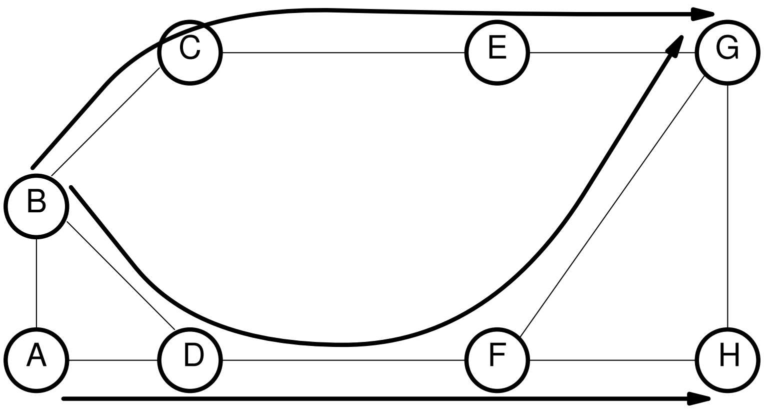

Figure 3.1: Avoiding collisions

A certain amount of communication is needed just to establish a consistent configuration. For example, in Figure 3.1, after the process of reconfiguration, if node A believes that a connection passes through node A and node B, and the local state on node B does not contain the same information, then the configuration is not consistent, and data transfer cannot take place correctly along the connection. This level of cooperation is taken care of at the connection management level. If we use an underlying

[1] Distributed algorithms may operate at any point on the trade-off curve, making them more convenient from the point of view of this research, where we are interested in exploring the trade-offs.

protocol such as RCAP or DCM, consistency of the configuration is ensured by the round-trip mechanism for channel establishment. However, even beyond this level of communication, opportunities exist for non-cooperative behavior between nodes. If node A and node B both decide to route a connection through link DF, and the resources on link DF are insufficient to support both, then one of them will fail at the admission control stage. This may even happen if sufficient resources for both channels exist on the link, due to the excess forward pass reservations of the first channel. If this information had been available to node B, then it could have chosen a different path (e.g., BCEG), on which this collision would not have occurred.

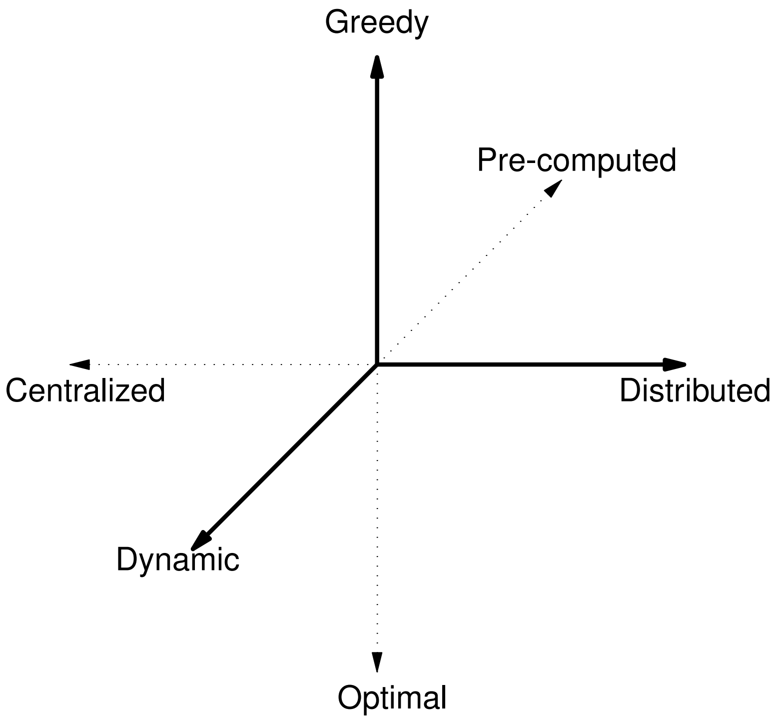

Figure 3.2: The sector of the solution space to be considered

Thus, the trade-off between speed and goodness of the solution computed by the distributed scheme needs to be explored. The best operating point for any network will depend on the network size, the communication latencies, the level of load on the network and many other variables. In addition, at present the details of the algorithm are completely unspecified. Thus, although we have restricted our solution space by considering only greedy, distributed, and dynamic solutions (see Figure 3.2), we are still left with a very large solution space to explore.

3.6. Issues not to be investigated

To reduce the size of the problem at hand, we will make certain simplifying assumptions about the solution. We have already decided not to explore fault detection and return to normal in any detail. Also, in the context of the Tenet scheme, the requirements of the channel management task can be satisfied by the services provided by the combination of RCAP and DCM. It fits all of the decisions taken so far, since it handles channels one at a time, using a distributed protocol to perform the admission control test, and does not require any pre-computation. Thus, we will also assume that channel management is performed by an existing realtime channel management protocol that supports route modification.

We will also not explore the mechanism by which the resource state information of the network, required by the routing algorithm, is distributed to all the nodes in the network. We assume that a reliable broadcast protocol, as described in Section 2.5.3, is used to maintain the routing table information. Of course, such a method implies that the tables are slightly out-of-date with respect to the true resource state of the network, and are updated as messages come in through the network. This also implies that more information can be obtained by simply waiting for the update messages to come in. Thus, there is a trade-off between the amount of information available on which routing decisions are based and the time that the scheme must wait to make the decisions.

When attempting to find a path to which a given channel may be rerouted, we have the same objectives as mentioned in Section 2.5.4, where we described the graph theoretic algorithm used to select a candidate route. The primary criteria are that sufficient resources exist to support the traffic parameters of the channel and that the end-to-end delay bound of the channel is satisfied. The secondary objective is to minimize the resource requirement of the candidate route. Since the route selection for the recovery process will also be carried out in the same greedy manner as before, we can use the Bellman-Ford algorithm, with certain modifications. These modifications are required to satisfy additional constraints, which are required by the recovery or tolerance schemes, and will be described as the constraints come up.

Thus, by fixing the channel management and route update protocols, and using the Bellman-Ford algorithm to find the min-hop path meeting the delay and resource requirements of a channel, we have simplified many aspects of the fault recovery activity. We will now describe the issues that remain to be investigated through simulation.

3.7. Remaining issues

The decisions made in the previous sections fix most of the variables in the scheme. We are left with a few more or less orthogonal factors: *the locus of reroute*, the *timing* of the rerouting attempt, the *retries* in the event of failure, and the use of *state prediction*. We will define these factors in this section.

For each of the factors named above, many solutions are feasible. Our objective is to explore the extremes of the range of possible approaches to each factor, as well as some intermediate solutions. Experimenting with the extremes will tell us if the factor being explored has any impact at all on the performance of the scheme as a whole. The experiments with the intermediate approaches will give us some idea of how the performance changes as the particular factor under consideration is varied. For example, if the performance according to some metric is roughly constant across all the approaches to a given factor, we can conclude that the particular metric is insensitive to variations within the factor. If the performance is best at one of the extremes, we can be reasonably sure that we do not need more careful experiments to search intermediate points for the best solution. If the performance of an intermediate approach to the factor is the best, more careful experiments are called for, perhaps in a more realistic environment involving an actual implementation, to find the best solution for that factor.

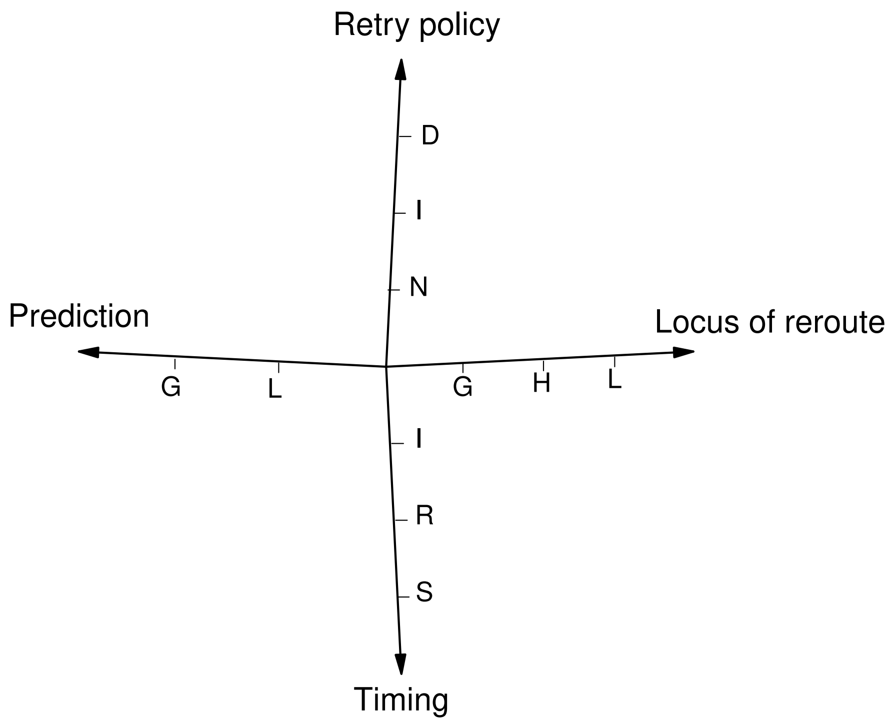

Figure 3.3: 4-dimensional solution space to be explored

We can think of the factors as defining a multi-dimensional space, and the various approaches as hyperplanes in which one coordinate is fixed. The schemes are points in the solution space, where all the coordinates are fixed by specifying an approach for each factor. In Figure 3.3, the approaches are marked on the dimensions by their abbreviated names (defined below). Not all the points in this 4-dimensional space correspond to reasonable schemes, as we shall see. Our simulation experiments will evaluate points in this space on multiple metrics of performance, attempting to find a scheme with good performance on all significant metrics.

3.7.1. Locus of reroute

The entire old route of the channel need not be subjected to a route selection process. In most cases, only one component of the channel, such as a link or a node, is affected by the failure, and it would suffice to find a new segment to replace a portion of the route. The locus of reroute determines the portion of the current route of the channel upon which a rerouting attempt will be made, and the node which performs the reroute selection. We consider three locus of reroute types: *Global reroutes* (G), *Local reroutes* (L), and *Hybrid reroutes* (H). In all types of locus of reroutes, the traffic and performance characteristics of any rerouted connection must be maintained after the rerouting is completed.

Global rerouting: With this approach, which corresponds to *end-to-end* rerouting (Section 1.4.2.1), the path of the rerouted connection is determined at the source node based on the current network load and the traffic and performance characteristics of the connection. This scheme considers all possible paths through the network. However, the source needs to have information about the resource state for all nodes in the network. The information on which this route selection is performed may be out of data, more so for nodes far from the source. This might make Global rerouting sensitive to the timing and prediction factors, which tune the level of information sharing and the time spent waiting for information.

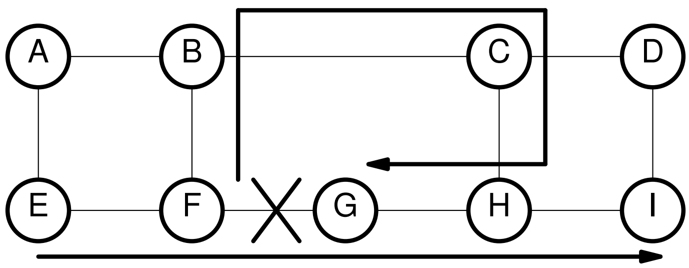

Figure 3.4: Local rerouting

Local rerouting: This approach to rerouting finds a new route from the node upstream of the fault to the node downstream of the fault. It combines this segment with the rest of the route from the original path. Since the path from the upstream node to the downstream node may include some links in common with the original path, such redundant sections are removed. For example, in Figure 3.4, the failed link FG is replaced by the segment FBCHG. However, on combining with the original path, the common portions GH and HG are removed, giving the new route EFBCHI. This process of route selection is performed at the fault detecting node. Since this is closer to the source of the fault, and the routing also takes place over the nodes in the immediate vicinity, Local rerouting should be able to route correctly with less waiting for information to come in. Thus, it should be less sensitive to the timing and prediction factors. The locality of the region affected might also make this scheme better able to handle multiple faults in different parts of the network, since one recovery attempt is relatively independent of the others. Finally, using as much of the old route as possible may also be beneficial, since the request is less likely to fail on these trunks, as some resources are already reserved.

Local rerouting is similar to *link* rerouting as discussed in Section 1.4.2.1. However, the establishment passes are carried out over the entire route, not just the new segment, for better resource balancing and to increase the chances of meeting the end-to-end delay requirements. This comes at the expense of a longer time to complete the rerouting, since the establishment passes must be made over the entire route.

Note that, although the need to perform end-to-end resource balancing does not allow us to get the shorter recovery time of link rerouting, the amount of traffic rerouted by Local rerouting should be as good as or better than that of link rerouting. Thus, to start with, we will explore Local rerouting as defined. Then, if the success of Local rerouting warrants it, we can explore the possibility of reducing the time to reroute by changing our resource balancing mechanism to exchange control messages over just the new segment.

Hybrid rerouting: Local rerouting considers a much smaller set of candidate routes than Global rerouting during route selection. This makes it much harder for Local rerouting to find a route that has sufficient resources. Hybrid rerouting attempts to retain the advantages of Local while mitigating its disadvantages, by first trying a Local reroute, then a Global reroute if the Local attempt fails.

3.7.2. Timing

The *reroute timing* policy selects the starting time of the reroute establishment attempt for each individual channel. This factor seeks to mitigate the effect of route collisions and database inconsistencies. *Route collisions* occur when an establishment request, during the forward pass along a link, consumes a large fraction of the remaining resources on the link, thus causing an immediately following forward pass establishment request to fail due to lack of resources, even though on the reverse pass the former establishment request would have relaxed its resource reservations so that the latter could have had its resource request honored. Reducing route collisions would increase the probability of establishing a connection. *Database inconsistencies* can also cause rerouting attempts within a small time interval of each other to interfere with each other. Had there been sufficient time between the two connection rerouting attempts for the route update protocol to exchange the necessary information, the routing database would have reflected a more accurate network load, and the latter of the two reroutes would have chosen a different path through the network.

Thus, by increasing the waiting time between rerouting attempts which use the same links in the network, we can improve the success of rerouting. Three possible approaches are considered to explore the effect of the timing factor: *Immediate* (I), *Random(n)* (R), and *Sequential* (S).

Immediate timing: With this timing approach, rerouting attempts are initiated by the controlling node as soon as a failure is reported to it. In the absence of any attempt to predict the behavior of other nodes, this approach is characterized by the least cooperation and potentially the most interference.

Random timing: With Random(n) timing, the rerouting attempt time is determined by generating a random value from a uniform distribution over an interval of duration n. This random value is added to the current time to obtain the reroute starting time. This randomization decreases the chances of two rerouting attempts being performed simultaneously over the same links in the network, and thus, should increase rerouting success. Different values of the randomization interval *n* yield different variations within Random timing.

Sequential timing: With this timing approach, all rerouting attempts are handled sequentially, with only one controlling node initiating a rerouting attempt at any given time. When all the reroute attempts of a single affected connection are completed, only then are rerouting attempts made on another connection. Sequential timing reduces route collisions and database inconsistencies. In fact, if we allow sufficient time between rerouting attempts, all nodes will perform route selection using up-to-date information. This is equivalent to a centralized scheme with respect to the rerouting success that can be achieved, but is obtained at the cost of a much higher average time to reroute. However, Sequential timing is useful since it represents the routing success that can be obtained by eliminating all collisions. We cannot expect to reroute a significantly larger amount of traffic without using global optimization techniques. As such, it serves as a good control experiment, against which we can measure the performance of the other approaches.

3.7.3. Retry policy

The *retry* policy determines the action that is taken if the route selection process fails, or if the establishment attempt along the selected path fails. We experimented with *No retries* (N), *Immediate retries* (I), and *Delayed retries* (I).

No retries: In the absence of retries, if the first attempt to find a route fails the channel is abandoned and the client is informed of the failure. The failed channel is torn down and the resources released immediately.

Immediate retries: With this approach to retries, if the route selection fails the channel is abandoned and torn down; if, however, the establishment attempt fails, new information provided by recent database update messages is incorporated, the saturated link (at which the previous reroute failed) is marked as a unusable for the route selection process, and a new route selection is performed immediately. An establishment is attempted on this route.

Delayed retries: With this approach, a retry is scheduled for a later time. The saturated link is not explicitly removed; rather, the route update information that comes in during the interval may cause the route selection to choose a different path. The method by which the interval is selected provides variations within this approach. We experimented with fixed intervals, exponentially increasing intervals, and randomization.

Retries are repeated on failure up to a pre-determined limit; different values of the limit give us different variations within each approach. We also experimented with using a cut-off time to stop further retries, instead of a number of attempts.

3.7.4. Prediction of network state

This factor determines the extent to which a node performing route computation modifies the information available in its routing tables at the time of route selection, on the basis of the predicted behavior of other nodes. The information in the routing tables models the actual resource state in the network but may be slightly out-of-date. Prediction aims to improve the accuracy of the modeling, to try and accurately determine the resource state likely to be found by the establishment request when it arrives at each intermediate node in the network. If prediction were perfect the effect would be to spread the different establishment attempts out onto different paths in the network, since the various nodes would cooperate to choose routes such that sufficient resources are available for each channel on each link.

Prediction has the same goal as the Random or Sequential timing approaches, where waiting for information to come in allows the correct routing decision to be made. Prediction attempts to obtain the same effect without the extra waiting time, by predicting the change in routing information which will occur.

We try two approaches within this factor: *Local prediction* (L) and *Global prediction* (G). These names (and abbreviations) are also used by the approaches for locus of reroute, but since we will always mention the factor being considered explicitly, no cause for confusion should arise. Since other intermediate prediction schemes that we can think of are fairly complicated, we decided to explore them only if the Global prediction approach offered sufficient improvement over the Local one. The two approaches are described below.

Local prediction: This is a minimal form of prediction, consisting of each node only trying to predict the result of its own establishment actions on its own local database. If a node performs two consecutive route computations, the latter is performed with the appropriate resources removed from the local database, so that it sees the state corresponding to what would exist after the forward pass of the former establishment request. This is the picture most likely to be seen by the second request as it propagates through the network after the first one. Also, in order to compute a reroute, the resources corresponding to the current channel are added to the network, since these resources are available to the alternate channel during the local admission tests. However, no attempt is made to make the behavior of the nodes globally predictable. This level of prediction requires no synchronization of reroutes, exchange of messages, or waiting time; thus, we make this the default mode of operation, and compare the performance of the Global approach against the performance of the Local one.

Global prediction: Although it is conceptually orthogonal to the other factors, Global prediction would work best with Immediate timing, since, with the other timing schemes, prediction is superfluous, as the information is already updated by waiting. In addition, the predictability of the timing of the Immediate approach makes the prediction scheme easier to implement, since we know that all the nodes are performing the route computations at roughly the same time. In contrast, the behavior of the Random timing approach is harder to predict, since the time at which the channel establishment is performed depends on the state of the pseudo-random number generation function. Thus, we only experiment with Global prediction in the context of the Immediate timing approach.

With Global prediction, all nodes wait for a sufficient interval of time without performing any channel administration activity, to ensure that the routing information has stabilized to the same value on each node. The fault message ensures that all nodes reach consensus about the set of channels to be rerouted. These channels are ordered using the same rule on all nodes. The route selection algorithm is performed for *all the channels* in the same order on all nodes. Since we only experiment with Global prediction in the context of Immediate timing, all of the nodes are running this algorithm at roughly the same time. After each route selection, the resources corresponding to the forward pass reservations are removed from the local routing tables. Since the same action is performed at all nodes, and the initial state of the routing tables are the same, all nodes make the same routing decisions. In fact, this is identical in effect to using a centralized routing algorithm. Thus, unless the assumptions of identical initial state and no extraneous channel management activity are broken, the routes found by this routing process will always have sufficient resources for the channel being rerouted, since the routing process models the actual establishment process. If the resource state during the forward pass might cause the establishment of any channel to fail on a given link, the algorithm should either find a different route for the channel or fail to find a route at the current time, since the resources used up by other channels are reflected in each local database.

Since the timing is Immediate, establishment attempts are generated at all the nodes at the same time. Thus, the resource states seen by the route selection process should be the same as the actual resources seen during establishment, and theoretically no attempt should fail during establishment. However, this might happen if our idealized assumptions about synchronization and complete coherence of the routing databases are not satisfied. It might also happen if the order of channel establishment is different from the order in the routing sequence, and the second channel has a tighter delay constraint than the first one.

Since the forward pass reservations are higher than those after relaxation, it is important to use retries for the channels for which routes could not be found during the first synchronized routing attempt. Retries for these channels, as well as for any channels which failed during establishment, may be performed in synchrony, after a sufficient interval to complete all establishment attempts and update all databases again. Alternatively, the retries may revert to the Random delay model. Both schemes were tried in our simulations.

It is important for Global prediction that, apart from the recovery action, all channel management activity (i.e., establishment of new channels) be frozen during the time that the synchronized routing attempts are being performed. Otherwise, the states seen by the different nodes will be different during the routing process, due to the change caused by this activity and the action of the route update protocol. This freezing of new establishments would not be intolerable if the time interval for the synchronized phase were short. Note that data delivery for already established channels that were not affected by the fault is not interrupted.

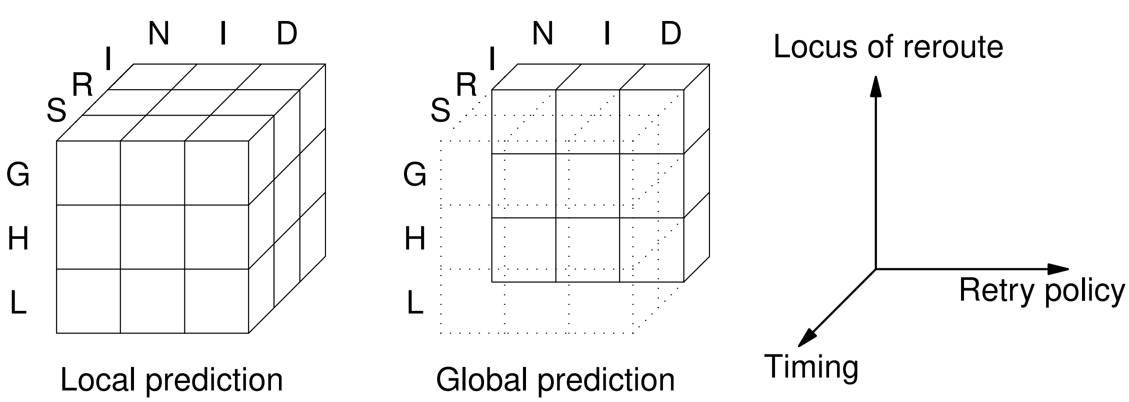

Figure 3.5: Valid regions of the 4-dimensional solution space

The Local prediction is valid with all combinations of the other factors. Global prediction, for the reasons mentioned above, is only useful with Immediate timing. Thus, the valid schemes in the 4-dimensional space lie in the regions shown in Figure 3.5, which shows the 4-dimensional space as two 3-dimensional spaces, one for Global prediction and the other for Local prediction. In the space for Local prediction, all the schemes are valid. These schemes are shown in solid lines. In the space for Global prediction, the regions containing the Random and Sequential timing approaches do not contain reasonable schemes. These regions are bounded by dotted lines.

3.8. Conclusion

In this chapter, we have formulated a framework for the systematic study of recovery of realtime channels. A promising portion of the solution space has been isolated by choosing greedy algorithms over ones which try to provide optimal solutions, distributed schemes over centralized control, and dynamic strategies rather than pre-computation methods. Figure 3.2 shows the sector of the solution space thus identified. We have chosen not to explore some issues by adopting the DCM protocol for changing the route of a channel, the Bellman-Ford algorithm to select the min-hop route meeting the performance and traffic constraints, and the reliable-broadcast-based route update protocol. We have also chosen not to address the problems of fault detection and return-to-normal, although we do return to them briefly in Chapter 5. The issues to be investigated are locus of reroute, reroute timing, network state prediction, and retry mechanisms. We have identified some possible approaches to each of these factors. The combination of these approaches define schemes, although not all of them are reasonable ones, as Figure 3.5 shows. In the next chapter, we proceed to explore the interesting schemes by simulation.